Building Digital Cities

by David Colleen
Planet 9 Studios, Inc.
San Francisco, California

draft March 12, 2002

Introduction

Computer simulation of realistic urban environments started about twenty years ago. In the past two years, computer game technology wedded to low cost advances in graphics hardware has lead to the ability to display high resolution, real-time cities on consumer level computers. The Internet has given us the means to share these datasets on a worldwide basis and to embed new functionality and linkages to other types of external data. We need an organized means to build and share these “digital cities”.

Purpose of this paper

For years there has been much discussion and development focused around 3D terrain simulation stemming from military simulation needs. Since military doctrine precluded fighting wars in cities, little effort was expended in simulating urban environments. The conflict in Somalia is often cited as the turning point as military planners realized that we would continue to be faced with the prospect of having to fight in urban zones. This coupled with growing concerns about global terrorism have caused a refocusing of efforts to simulate our cities.

It is the purpose of this paper to set out techniques for building high performance 3D urban environments as well as to suggest some standards to aid in data her. I should note that there is another arena of city building revolving around automated collection techniques using stereo pair photography, laser scanning, etc. While those techniques show promise, they do not yet yield sufficient fidelity for most of our uses. This paper will instead focus on high precision hand modeling techniques that have been the province of architects and computer game makers.

Assumptions

For the purposes of this paper, I will assume that the reader has a good understanding of the Internet and has a working knowledge ob 3D tools and concepts. While there are many fine 3D authoring tools and real-time display formats, for the purposes of this paper, I will refer to authoring in 3D Studio Max (the leading 3D authoring tool), metadata storage in XML and VRML 97 (the ISO standard for real-time 3D on the Internet).

Building Real Time City Models

In creating an urban dataset, there are some clear objectives:

- Create the highest sense of realism while,
- Maintaining an acceptable frame rate that,
- Uses a data structure that is comprehendible by others and that,
- Can be added to by others and that is,
- Compact in size for transmission via the Internet.

When we first began building city models for Internet use, we limited the files to 2,500 polygons. Today with improvements in hardware, software and Internet transmission speeds, we regularly use models over 100,000 polygons. Each year we find that rules for model building change with rapid advances in technology. Each rendering platform also has strengths and weaknesses that can be exploited with careful model building and data structuring. In spite of the difficulties in addressing a “moving target” topic I will offer some useful generalities for real-time model building.

Pre-Planning

Each model building project should begin with a design document that states the project objectives, design criteria and technical constraints. Some of the questions that should be addressed are:

- What is the polygon budget?
- What is the design platform?
- What is the texture budget?
- Who are the target users?
- What are the file size constraints?
- What is the design screen resolution?
- What is the rendering platform?

Here is an example of a typical polygon budget:

1. Ground Plane 10,000 polygons
2. Buildings 40,000 polygons (40 buildings @ 1,000 polygons / bldg. avg.)
3. Street Furniture 6,000 polygons (100 objects @ 60 polygons / object)
4. Landscaping 4,000 polygons (1,000 trees @ 4 polygons / tree)

This totals 60,000 polygons. Often polygon budgets get more detailed than the example given.

Planning For Online Use

Deploying an urban dataset over the Internet requires careful consideration of user bandwidth constraints and server responsiveness. It is important to understand what are practical wait times for a user and how this translates into workable file sizes and the quantity of textures to be used. In the days that we designed for users on 28.8 dial-up connections we had an internal rule that dictated that no geometry file was to exceed 50k, no texture file could exceed 11k and there was to be a maximum of 20 textures used. These standards seem antiquated now except that as we are addressing technical limitations for hand held computers, we find ourselves looking back to the older limits. In designing for T1 / DSL users we have raised our limits to about 300k for geometry files, 60k for texture files with as many as 100 textures.

Compression of geometry files is critical to maintaining small file sizes. VRML supports gzip compression with no difference in performance. Compression rates vary, but we typically see 5:1 compression rates. It also helps to reduce file sizes with the help of an optimization tool such as Chisel.

The overall number of textures used is significant relative to the overall texture memory used but also for the number of “hits” that the server must address. It is helpful to try to reduce the overall number of textures in a project but also to collage textures together into larger textures to reduce the load on the server and to reduce the memory footprint on the client computer. (See the section on texture collaging below.)

Low Polygon Geometry

Good Polygons

Cracks, Shared Vertex’s and Decimal Places

“T” Vertex’s

Quads and N-Gons

Some older 3D file formats define a polygon as having only three vertexes. Most modern formats support polygons with four or more vertexes. It is more efficient to render a surface containing 10 four sided polygons rather than 20 three sided polygons. The file size is also slightly smaller. Where possible, quads and n-gons (polygons with five or more vertexes) should be used.

Texturing to Simulate Detail

In the earliest days of real-time simulation, polygon budgets were extremely low forcing modelers to rely on texturing to simulate detail that could not be built in 3D. Texture resolutions were also limited due to download and display constraints. The tendency was to

use a single texture map for each side of a building. The limitations of this approach became apparent as the viewer approached the building and saw enormous pixels.

Today, using modern engines, we are seeing polygon budgets for individual buildings between 500 and 2,000 polygons and textures as large as 2048 x 2048. This has allowed us to include medium sized building details and to begin breaking down facades into repitious panels. For instance, a building with a repeated window panel may be comprised of four texture variants randomly sprinkled over the buildings surface to achieve a more natural appearance.. Sice individual textures cover a relatively small area, pixilation problems are minimized. We are even occasionally adding individual buildings to scenes with polygon counts in the 50,000 to 300,000 range.

Pre-Lit Texturing

Mip-Mapping

Mip-mapping is a technique used to replace texture maps with progressively lower resolution textures as the view moves away from the textured object. This reduces the overall memory footprint of the scene and consequently increases the frame rate. Modern graphics cards perform this function automatically if the original texture adheres to mip-map texture dimensions. Mip-mapable dimensions are based on the power of 2 such as 16, 32, 64, 128, 256, 512, etc. With most rendering engines, textures can be rectangular.

It is often a difficult conceptual leap to take a pristine building façade texture map and modify the overall aspect ratio to conform to mip-map dimensions. (i.e. A 640 x 480 texture is re-sized to 512 x 512.) It helps to remember that the rendering engine is already distorting textures to appear in perspective. The additional distortion required to resize a texture to mip-map dimensions yields no visible image degradation and significant performance improvement.

Collaging

Memory footprint … download time

Texture Compression

Color by Vertex

Minimizing Hidden Geometry

Using Normals to Advantage

By definition, polygons are opaque on one side and transparent on the other. Most of our data represents a shell so it is OK to be transparent on the side that will never be viewed. Occasionally, we may wish to have a polygon be opaque on both sides. This can me

achieved by using the “solid FALSE” tag within the VRML IndexedFaceSet description or by using a “2-sided material” if authoring in Studio Max. Making a polygon opaque on both sides doubles the rendering load for that object, so please use this feature sparingly.

Imposters

Imposters are flat, texture mapped polygons that are meant to represent a more detailed object. There are several types of imposters; scrims, billboards and x-objects. A scrim is a background diorama that represents objects in the distance. Scrims may represent foliage, a city skyline or a store interior. Billboards are flat, textured polygons that rotate to always face the viewer. This is useful for trees and statues. X-objects are usually two or three intersecting planes used to represent trees (“x-trees”). A few systems also support dynamically generated imposters to replace distant geometry. This is not yet a mainstream approach.

Levels of Detail (LOD’s)

In the earliest days of simulation, display platforms were highly constrained by the overall number of polygons within the view frustum. LOD systems were invented to counteract this limitation by using multiple models of the same object, built at different resolutions. With LOD’s, progressively more detailed models are displayed as the viewer gets closer to an object. This has the benefit of reducing the overall scene polygon count. It also has the negative aspect of increasing the overall file size and increasing the authoring time. In recent times, with more robust display platforms, we have actually noticed performance reductions in models using LOD’s. I encourage the author to become familiar with the strengths and weaknesses of the rendering engine to be used.

A new generation of LOD type geometry systems has appeared called multi-resolution geometries. Multi-resolution geometries work in two ways. Multi-resolution geometries can automatically reduce the resolution of distant objects to maintain a consistent frame rate or polygon count within the scene. Well done systems are able to maintain proper texturing and object profiles without the appearance of seams, swimming or popping as is inherent in traditional LOD schemes. The other way that multi-resolution geometries are used is with streaming systems online.. In this scheme, a low-resolution proxy is initially loaded in the scene. Progressively higher resolution is displayed over time as additional detail is downloaded. VRML viewer developers are just beginning to address the use of these technologies. Eventually, they may replace traditional LOD schemes altogether.

Elevation Grids

Primitives

Many programmers feel that objects define by primitive shapes, such as cubes, spheres, cones, etc., run faster than polygonal shapes. While using primitives saves somewhat in file size, our bench tests show primitives to run significantly more slowly than their polygonal

counterparts.

Extrusions

Indexed Line Sets

Collision

Data Structure

Scene Management Techniques

Data Optimization

Publishing the Data

Summary

Future Topics to Consider:

1. Data Validation.
2. Date tagging and time lapse simulation.
3. Automated collection techniques.
4. Needed areas of future technology development.
5.

Appendix A

A Proposed Resolution Classification System

Several years ago, the military developed a classification system for the resolution of terrain data called DTED (digital terrain elevation data). To date, no standards have evolved to classify urban data. There are several pertinent aspects that characterize the resolution of an urban dataset, such as texture resolution, building detail and overall scene detail. I would like to propose the following classification system based on DCD (digital city data):

Item	**DCD0**	**DCD1**	**DCD2**	**DCD3**	**DCD4**
Texture Resolution	1m.	50cm.	25cm.	10cm.	5cm.
Building Detail	10m.	5m.	1m.	50cm.	10cm.
Street Detail	Flat	Curbs	Curbs	Curbs	Curbs & Cuts
Landscaping Detail					
Trees	Billboard	X-Trees	X-Trees	X-Trees w/Trunks	Polygonal
Shrubs			Scrim	X-Bush	xx

Shrubs			Scrim	X-Bush	xx
Topo Features			Slight	xx	xx
Pathways			Major	All	All w/ Curbs
Fountains					
Statues		Billboard	Billbd. w/Base	Polygonal	Polygonal
Walls & Steps		Basic	W/ Ramps	W/ Ramps	W/ Steps
Street Furniture Detail					
Traffic Signs			Yes	Yes	Yes
Traffic Signals		Yes	Yes	Yes	Yes
Street Signs		Generic	Readable	Readable	Readable
Waste Receptacles				Yes	Yes
Benches				Yes	Yes
News Stands			Yes	Yes	Yes
News Boxes				Yes	Yes
Street Lights		Yes	Yes	Yes	Yes

Texture resolution is stated in meters describing the approximate size of a pixel displayed on a building façade.

Building detail refers to the size or granularity of details modeled. As an example, DCD0 buildings would be basic boxes; DCD1 buildings would include setbacks and stepping; DCD2 buildings would add columns and mechanical penthouses; DCD3 buildings would have cornices and bay windows and DCD4 buildings would have window mullions and recessed doorways.

Appendix B

Planet 9 Studios Naming & File Structure System

Object Naming - Most geography based display systems have been built on cells defined by latitude and longitude. Our business needs over the years have been focused on cities and their buildings. It would have been possible for us to use a lat/lon convention for the naming and structuring of our cities but it would have been quite un-intuitive and difficult to implement. Instead, we choose to develop a system based on place names and property descriptions. Here is an example:

- California (state)
- Bay Area (region)
- San Francisco (city)
- Financial District (neighborhood)
- Block # (city designation)

- Lot # (parcel)

As with most structured systems, there is a need to handle exceptions. For instance, in San Francisco, the waterfront buildings located on piers do not have lot and block numbers. In this case, we have simply named objects by their well know names such as “Pier 45”.

Texture naming – Our texture naming is directly derived from the Object Naming scheme above. A typical texture name would look like this: 123456w.jpg where 1234 = block, 56, = lot and “w” is a modifier, in this case indicating a west facing elevation. Here are some more of our standard modifiers:

- n, s, e, w, = north, south, east and west facing elevations.
- na, nb = this would indicate multiple north facing elevations.
- x = transparency map
- b = bump map
- e = environment map
- s = shininess map
- i = illumination map

Increasingly, real-time engines are supporting multi-texturing. We may find new mapping types over time. There are also novel uses of existing texture formats such as using the alpha channel for illumination information rather than for transparency. It is also common to collage texture together to improve performance. In these cases we often combine the textures for an entire block into one texture. In this case we use the block number alone as the texture name.

Case Sensitivity -

Data Hierarchy -

Suggested Reading

Online Resources

Glossary